In The Specification:

In the Description of the Related Art at Page 2, Line 24, please replace the paragraph beginning with "Various attempts have been ..." with the following:

Various attempts have been made to mitigate this problem. EP-962860-A describes a process whereby one JVM can fork into a parent and a child process, this being quicker than setting up a fresh JVM. Another approach is described in "Oracle JServer Scalability and Performance" by Jeremy Litzt, July 1999 ~~(see: http:www.oracle.com/database/documents/jserver_scalability_and_performance_twp.pdf).~~ The JServer product available from Oracle Corporation, USA, supports the concept of multiple sessions (a session effectively representing a transaction or application), each session including a JServer session. Resources such as read-only bytecode information are shared between the various sessions, but each individual session appears to its JServer client to be a dedicated conventional JVM. A parallel JVM approach is also described in "Building a Java virtual machine for server applications: the JVM on OS/390" by Dillenberger et al, IBM Systems Journal, Vol 39/1, January 2000. Here two types of JVM are utilised, a resource-owning JVM which loads and resolves necessary system classes, and subsequent "worker" JVMs which can reuse the resolved classes.

In the Description of the Related Art at Page 3, Line 19, please replace the paragraph beginning with "The above systems generally ..." with the following:

The above systems generally require significant changes to the underlying JVM, and this can in turn lead to different problems in terms of portability and so on (they must be implemented on each JVM platform). Therefore a somewhat different approach is adopted by the Server Side Java Technology from The Santa Cruz Operation Inc (SCO) ~~(see http://www.sco.com/10xmore/info/index_and_related_links)~~. This teaches a layer between applications and the JVM that allows multiple applications to run unchanged as separate processes on a single JVM. Amongst other things this improves scalability, because there is no

need to provide system memory, perform garbage collection, etc for multiple JVMs.

In the Description of the Related Art at Page 4, Line 15, please replace the paragraph beginning with "A further development of this ..." with the following:

A further development of this approach is represented by Echidna~~, described at and available from http://www.javagroup.org/echidna~~. As with the SCO Server Side Java, this is a class library which allows multiple applications to run as different threads within a single JVM, rather than having to spawn a new JVM for each application. This gives much shorter start-up times because system classes do not need to be loaded into each new JVM, rather they are available for each thread (i.e. application). Also because Echidna is written as a Java application, it is automatically portable and will work with any JVM (or JRE).

In the Detailed Description at Page 21, Line 8, please replace the paragraph beginning with "To actually start the execution ..." with the following:

To actually start the execution of the application, the LRE uses the Java Reflection API to find the public "main" method of the initial class, (see Sun Microsystems, Inc. Reflection Documentation, 1997~~, at http://java.sun.com/products/jdk/1.1/docs/guide/reflection/~~). The LRE then calls Method.invoke() on this, with any required parameters, in the run() method of the thread created for the application. Execution then proceeds in this thread, leaving the LRE free to load and begin execution of any other applications.

In the Detailed Description at Page 29, Line 8, please replace the paragraph beginning with "Parsing a Java class file in byte code ..." with the following:

Parsing a Java class file in byte code form for particular instructions is relatively straightforward due to the standard formalised definition of byte code contents in order to run on the JVM (see in particular chapter 4 of the above-referenced book by Lindholm and Yellin). The same applies to altering the byte codes to perform simple modifications, such as deleting a method, or adding a method which calls back to LRE (note that such changes will also require updating to the constant pool to delete or add the relevant method name and any other associated constants). In addition, there are also tools available to help with Java bytecode manipulation and/or decompilation. One such tool for bytecode manipulation is Jikes ~~(available from http://www.alphaworks.ibm.com/tech/jikesbt),~~ which allows searching for particular instructions in a class and updating such instructions, etc, and also takes care of the necessary interaction with the constant pool. Accordingly bytecode manipulation unit 446 may optionally use some existing tool for this purpose (nb the current version of Jikes in fact requires a class name rather than a class file as input, in other words also performs the operation of unit 444 as well, but clearly this is not a key part of its functionality and could be easily modified.